

SEMANTIC WEB TECHNOLOGIES I

Lehrveranstaltung im WS12/13

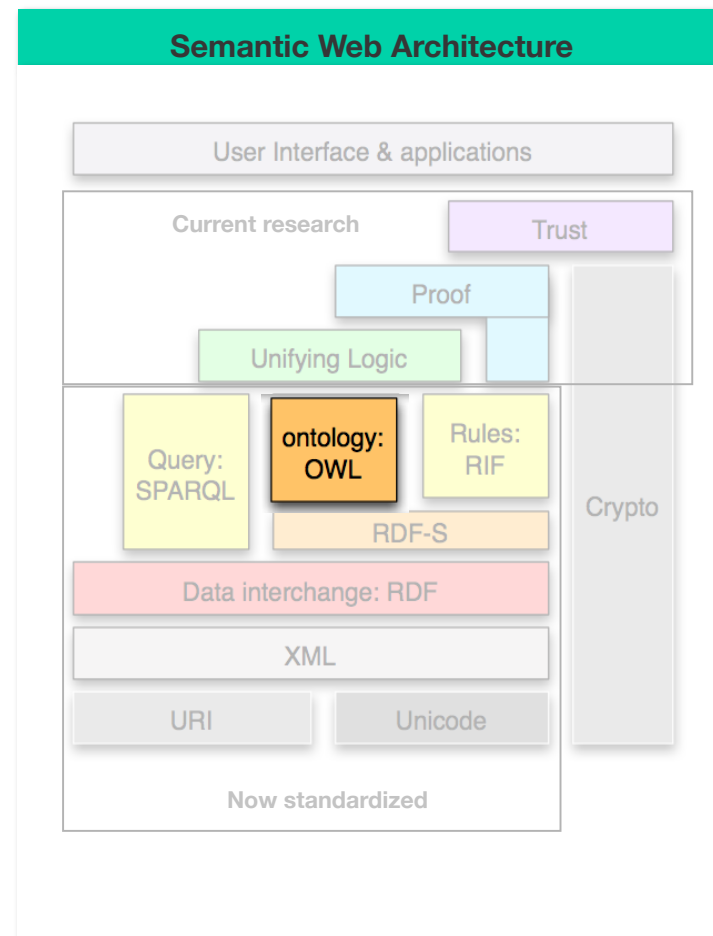
PD Dr. Sebastian Rudolph

Dr. Thanh Tran

OWL – SYNTAX & INTUITION

2/2

PD Dr. Sebastian Rudolph



Agenda

- + Anmerkungen zur RDF-basierten Syntax von OWL
- + fortgeschrittene Modellierungsmittel in OWL
 - mehr Klassenkonstruktoren
 - erweiterte Modellierungsmöglichkeiten für Properties
 - Handhabung von Datenwerten
 - Profile von OWL

RDF-BASIERTE OWL-SYNTAX



```
@prefix :      <http://www.example.org/#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

:owns          rdf:type owl:ObjectProperty .
:caresFor       rdf:type owl:ObjectProperty .
:Cat            rdf:type owl:Class .
:Dead           rdf:type owl:Class .
:Alive          rdf:type owl:Class .
:Healthy        rdf:type owl:Class .
:HappyCatOwner rdf:type owl:Class .

:owns           rdfs:subPropertyOf :caresFor .

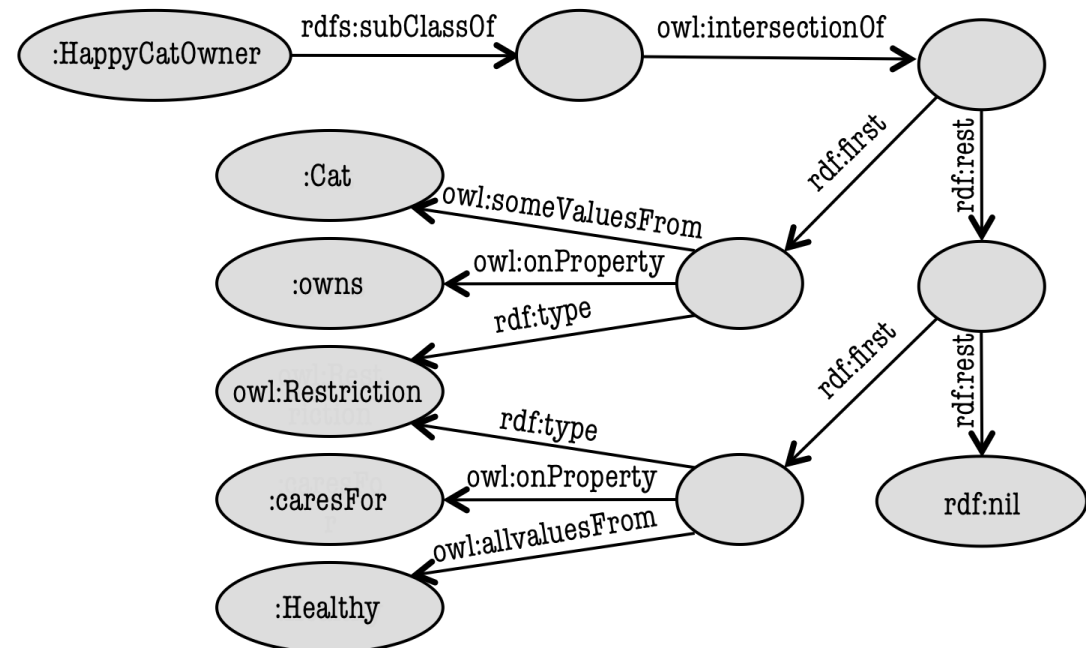
:Healthy        rdfs:subClassOf [ owl:complementOf :Dead ] .
:Cat            rdfs:subClassOf [ owl:unionOf (:Dead :Alive) ] .
:HappyCatOwner  rdfs:subClassOf
  [ owl:intersectionOf
    ( [ rdf:type owl:Restriction ;
        owl:onProperty :owns ; owl:someValuesFrom :Cat ]
      [ rdf:type owl:Restriction ;
        owl:onProperty :caresFor ; owl:allValuesFrom :Healthy ] )
  ] .

:schrödinger    rdf:type :HappyCatOwner .
```

RDF-BASIERTE OWL-SYNTAX

APIER

```
:HappyCatOwner rdfs:subClassOf
  [ owl:intersectionOf
    ( [ rdf:type owl:Restriction ;
        owl:onProperty :owns ; owl:someValuesFrom :Cat ]
      [ rdf:type owl:Restriction ;
        owl:onProperty :caresFor ; owl:allValuesFrom :Healthy ] )
  ] .
```

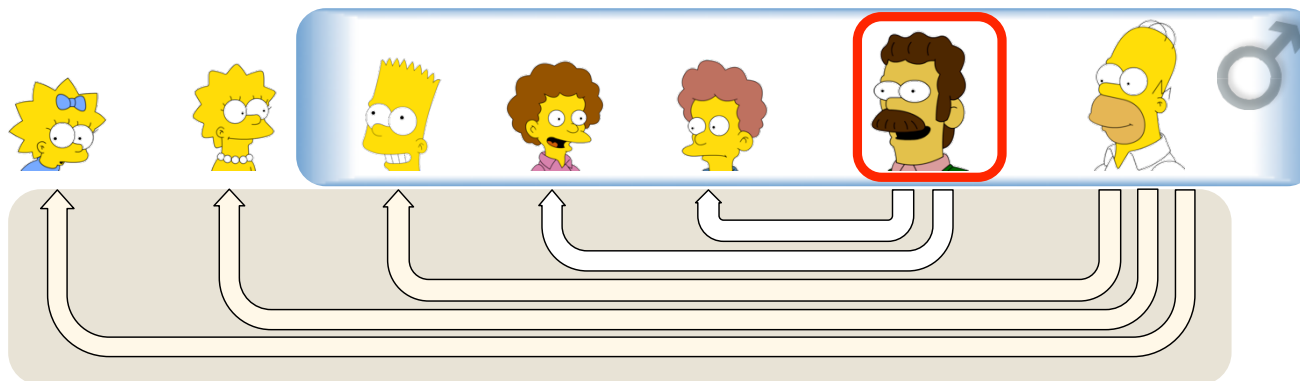


Mehr komplexe Klassen: Qualifizierte Kardinalitätsrestriktionen

AIFB 

- ```
[rdf:type owl:Restriction ;
 owl:minQualifiedCardinality
 "n"^^xsd:nonNegativeInteger ;
 owl:onProperty prop; owl:onClass class]
```
- Beispiel:  

```
[rdf:type owl:Restriction ; owl:minQualifiedCardinality
 "2"^^xsd:nonNegativeInteger ;
 owl:onClass ex:Male; owl:onProperty ex:parentOf]
```



# Mehr komplexe Klassen: Qualifizierte Kardinalitätsrestriktionen

AIFB 

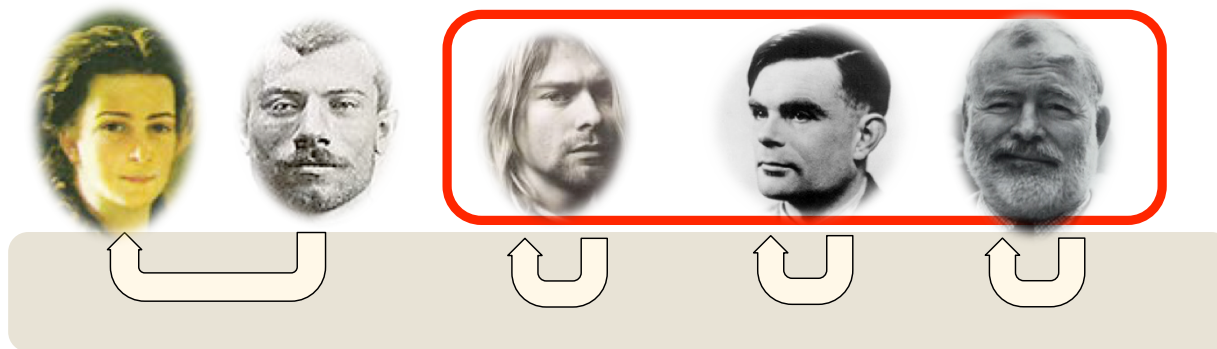
- analog zu `owl:minQualifiedCardinality` („mindestens“):
  - „höchstens“:  
`owl:maxQualifiedCardinality`
  - „genau“:  
`owl:QualifiedCardinality`

# Mehr komplexe Klassen: Self-Restriktion

AIFB 

- |   |                             |                                    |
|---|-----------------------------|------------------------------------|
| [ | <code>rdf:type</code>       | <code>owl:Restriction ;</code>     |
|   | <code>owl:onProperty</code> | <code>prop ;</code>                |
|   | <code>owl:hasSelf</code>    | <code>"true"^^xsd:boolean ]</code> |
  
- Beispiel: 

|   |                             |                                   |
|---|-----------------------------|-----------------------------------|
| [ | <code>rdf:type</code>       | <code>owl:Restriction ;</code>    |
|   | <code>owl:onProperty</code> | <code>ex:hasKilled ;</code>       |
|   | <code>owl:hasSelf</code>    | <code>"true"^^xsd:boolean]</code> |





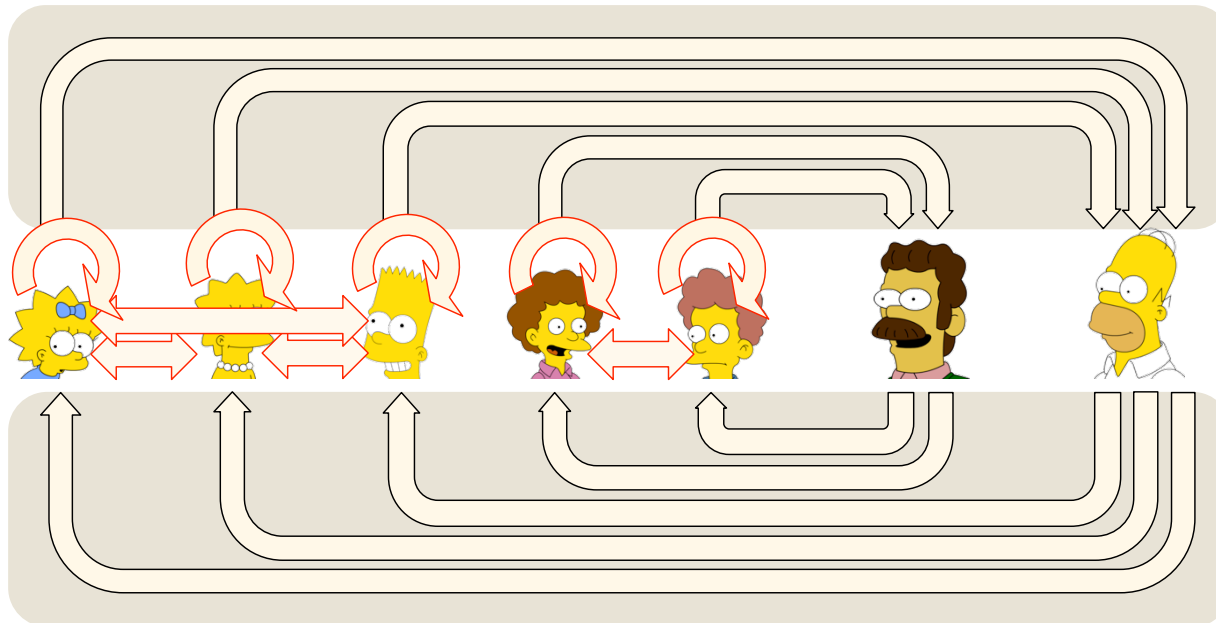
# Property Chain Axioms

AIFB 

- *prop* `owl:propertyChainAxiom` ( *prop1*, ... , *propn* ) .

- Beispiel:

`ex:siblingOf owl:propertyChainAxiom`  
`( ex:childOf, ex:parentOf ) .`



# Probleme mit Entscheidbarkeit



- role chain axioms können leicht zur Unentscheidbarkeit führen
- um Entscheidbarkeit zu garantieren, werden zwei globale Einschränkungen für OWL-DL-Ontologien gefordert:
  - die Menge der property chain axioms und subproperty Aussagen muss *regulär* sein
  - Properties, die in Kardinalitäts- und Self-Restriktionen verwendet werden müssen *simple* Properties sein

# Property-Chain-Axiome: Regularität

AIFB 

- wir kürzen ab:  
 $R \text{ owl:propertyChainAxiom } (S_1 \dots S_n).$  mit  $S_1 \circ \dots \circ S_n \sqsubseteq R$   
 $S \text{ owl:subPropertyOf } R.$  mit  $S \sqsubseteq R$
- Regularitätsbedingung: es muss eine lineare Ordnung  $<$  auf den Properties existieren, so dass jedes Property-Chain-Axiom und jedes Subproperty-Axiom eine der folgenden Formen hat (wobei  $S_i < R$  für alle  $i = 1, 2, \dots, n$  gelten muss):

$$R \circ R \sqsubseteq R \quad [\text{owl:inverseOf } R] \sqsubseteq R \quad S_1 \circ S_2 \circ \dots \circ S_n \sqsubseteq R$$

$$R \circ S_1 \circ S_2 \circ \dots \circ S_n \sqsubseteq R \quad S_1 \circ S_2 \circ \dots \circ S_n \circ R \sqsubseteq R$$

- Beispiel 1:  $R \circ S \sqsubseteq R \quad S \circ S \sqsubseteq S \quad R \circ S \circ R \sqsubseteq T$   
regulär mit Ordnung  $S < R < T$
- Beispiel 2:  $R \circ T \circ S \sqsubseteq T$   
nicht regulär, da Form nicht zulässig
- Example 3:  $R \circ S \sqsubseteq S \quad S \circ R \sqsubseteq R$   
nicht regulär, weil keine passende Ordnung existiert

# Property-Chain-Axiome: Simplizität

AIFB 

- Kombination von Property-Chain-Axiomsen und Kardinalitäts- oder Self-Restriktionen kann zur Unentscheidbarkeit führen
- Bedingung: verwende nur *simple* Properties in Kardinalitäts- oder Self-Restriktionen (d.h. solche, die nicht – direkt oder indirekt – von Property-Ketten abgeleitet werden können)
- formal:
  - für jedes Property-Chain-Axiom  $S_1 \circ S_2 \circ \dots \circ S_n \sqsubseteq R$  mit  $n > 1$ , ist  $R$  non-simpel
  - for jedes Subproperty-Axiom  $S \sqsubseteq R$  mit non-simplem  $S$  ist auch  $R$  non-simpel
  - alle anderen Properties sind simpel
- Beispiel:
 

$Q \circ P \sqsubseteq R$   
 non-simpel:  $R, S$

$R \circ P \sqsubseteq R$   
 simpel:  $P, Q$

$R \sqsubseteq S$

$P \sqsubseteq R$

$Q \sqsubseteq S$

# Eigenschaften von Properties

AIFB 

- mit OWL kann auch spezifiziert werden, dass Properties die folgenden Eigenschaften haben:

- disjoint from another
- functional
- inverse functional
- transitive
- symmetric
- asymmetric
- reflexive
- irreflexive

„syntaktischer Zucker“  
(mit schon eingeführten  
Features ausdrückbar)

# Datentypen in OWL

AIFB 

- wie in RDF, können mithilfe von Properties Individuen mit Datenwerten verknüpft werden:

`ex:john ex:hasAge "42"^^xsd:integer .`

# RANGES VON DATENTYPEN

## AIFB

- Ranges von konkreten Rollen:  
Datentypen (häufig verwendet: Datentypen aus XML Schema)
- Beispiel:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
...
ex:hasAge rdfs:range xsd:integer .
```


- Interpretation der Datentypen ist in XML Schema definiert (OWL beseitigt einige Unklarheiten, z.B. “Gibt es eine Überschneidung der Wertebereiche von floating point und integer?”)
- Achtung: Datentypen müssen immer noch explizit angegeben werden! Mit dem obigen Axiom erhält man:

```
ex:jean ex:hasAge "17"^^xsd:integer . ← Korrekt
ex:paul ex:hasAge "23"^^xsd:decimal . ← Korrekt
ex:claire ex:hasAge "42" . ← Inkonsistent!
```

# DEFINIEREN NEUER DATENTYPEN

AIFB 

- in XML Schema können Datentypen eingeschränkt werden  
→ **datatype facets**
- Beispiel:



```
ex:personAge owl:equivalentClass
 [rdf:type rdfs:Datatype;
 owl:onDatatype xsd:integer;
 owl:withRestrictions (
 [xsd:minInclusive "0"^^xsd:integer]
 [xsd:maxInclusive "150"^^xsd:integer]
)
] .
```

- die möglichen Arten der Einschränkung hängen vom Datentyp ab, einige facets sind nur eingeschränkt in OWL verfügbar → für Details s. Specs



# EINFACHE DATENINTEGRATION IN OWL

AIFB 

- praktisches Problem: gegeben Ontologien aus verschiedenen Quellen, welche URIs beziehen sich auf dieselben Individuen?
- typischer Ansatz in OWL:
  - Gleichheit explizit machen durch `owl:sameAs`
  - Definition von Properties als invers funktional (“selber Wert → selbes Individuum”)
- Probleme:
  - `owl:sameAs` erfordert explizite Mappings (selten im Web)
  - OWL DL verbietet invers funktionale konkrete Rollen
  - Es kann nur eine Property zur globalen Identifikation verwendet werden, keine Kombinationen (zum Beispiel ist nicht darstellbar: “Alle Teilnehmer von SWebT1 mit demselben Namen und dem selben Geburtstag sind gleich.”)

# OWL 2 KEYS



AIFB 

- OWL 2 stellt eine Möglichkeit bereit, das zu modellieren:

```
ex:SwebT1Student owl:hasKey (ex:name, ex:birthday) .
```

- **Einschränkung:** Keys beziehen sich nur auf benannte Individuen, d.h. Objekte der Interpretationsdomäne, für die es eine URI gibt.
- Konkret: wenn es zwei URIs  $u$  und  $v$  gibt und ein Name  $n$  und ein Datum  $b$  existiert, so dass

```
u rdf:type ex:ESSLLISTudent; ex:name n ; ex:birthday b .
v rdf:type ex:ESSLLISTudent; ex:name n ; ex:birthday b .
```

dann können wir schließen:  $u$  owl:sameAs  $v$  .

# OWL 2 PROFILE



AIFB 

- **Design-Kriterium für OWL “tractable” Profile:**  
Identifikation maximaler OWL 2 Teilsprachen für die automatisches Schlussfolgern in PTime realisiert werden kann.
  - Hauptquelle für “intractability”: **Nichtdeterminismus** (erfordert Raten/ Backtracking)
  - `owl:unionOf`, **oder** `owl:complementOf` + `owl:intersectionOf`
  - Max.-Kardinalitäts-Restriktionen
  - Kombination von existentiellen (`owl:someValuesFrom`) and universellen (`owl:allValuesFrom`) Restriktionen in Überklassen
  - Aufzählungsklassen (`owl:oneOf`) mit mehr als einem Element
  - endliche Datentypen
- diese Features sind in keinem OWL 2 Profil erlaubt

- **OWL-Profile basierend auf der Beschreibungslogik EL++**
- Intuition: Konzentration auf terminologische Ausdruckstärke für leichtgewichtige Ontologien
- erlaubt `owl:someValuesFrom` aber nicht `owl:allvaluesFrom`
- Domains, Klassen-/Property-Hierarchien, `owl:intersectionOf`, disjunkte Klassen/Properties, property chains, `owl:hasSelf`, `owl:hasValue`, Keys
- keine inversen oder symmetrischen Properties
- `rdfs:range` erlaubt aber mit Einschränkungen
- verboten: `owl:unionOf`, `owl:complementOf`
- zahlreiche Einschränkungen auf erlaubten Datentypen

# OWL 2 EL: FEATURES

AIFB 

- Standard reasoning in OWL 2 EL:  
PTime-vollständig
- wird verwendet in praktisch relevanten Ontologien:  
primäres Beispiel ist SNOMED CT  
(Ontologie zu klinischem Vokabular mit Klassen und Properties  
in der Größenordnung  $10^5$ )
- schnelle Implementierungen sind verfügbar:  
volle Klassifikation von SNOMED-CT in <10min;  
Echtzeitverhalten, wenn adäquat vorverarbeitet (Aufteilung in  
“logische Module”)

# OWL 2 QL



- **OWL-Profil zugeschnitten auf datenintensive Anwendungen**
- Intuition: Ontologie wird in Datenbank gespeichert, Verwendung von OWL-Klassen als leichtgewichtige Anfragen, Anfragebeantwortung erfolgt durch Umwandlung in und Ausführung von SQL
- verschiedene Einschränkungen bei Unter- und Überklassen in `rdfs:subClassOf`:
  - Unterklassen: Klassennamen oder `owl:someValuesFrom` (existential) mit `owl:Thing`
  - Überklassen: Klassennamen, `owl:someValuesFrom` oder `owl:intersectionOf` with Überklassen als Argument (rekursiv), oder `owl:complementOf` mit Unterklassenargument
- Property-Hierarchien, Disjunktheit, Inverse, (A)symmetry zulässig, Range und Domain eingeschränkt
- disjunkte Klassen und Klassenäquivalenz nur für Unterklassenargumente (s.o.)
- **verboten:** `owl:unionOf`, `owl:allValuesFrom`, `owl:hasSelf`, `owl:hasKey`, `owl:hasValue`, `owl:oneOf`, `owl:sameAs`, `owl:propertyChainAxiom`, `owl:TransitiveProperty`, **Kardinalitäten, (invers) funktionale Properties**

# OWL 2 QL: FEATURES

AIFB 

- Standard reasoning in OWL 2 QL: PTime  
für spezielle assertionale Anfragen sogar LogSpace  
(das ist besser als PTime)
- bequeme leichtgewichtige Schnittstelle für Legacy-Daten
- schnelle Implementierungen aufsetzend auf bestehenden Datenbanksystemen (relational or RDF-basiert): skaliert auf sehr große Datenmengen

# OWL 2 RL



- **OWL-Profil, welches in OWL ausdrückbare (Horn-)Regeln abdeckt:**
- Intuition: Subklassenaxiome in OWL RL können als Implikationen verstanden werden mit Regelkopf (Überklasse) und Regelrumpf (Unterklasse)
- verschiedene Einschränkungen für Unter und Überklassen bei `rdfs:subClassOf`:
  - Unterklassen: Klassennamen, `owl:oneOf`, `owl:hasValue`, `owl:intersectionOf`, `owl:unionOf`, `owl:someValuesFrom` mit zulässiger Unterklasse als Argument
  - Überklassen: Klassennamen, `owl:allValuesFrom` oder `owl:hasValue`; max. Kardinalitäten nur mit 0 oder 1 zulässig, alle mit zulässiger Überklasse als Argument
- Domains und Ranges nur für zulässige Unterklassen; Property-Hierarchien, Disjunktheit, Inverse, (A)symmetrie, Transitivität, role chains, (inverse) Functionalität, Irreflexivität voll unterstützt
- Disjunktheit und Keys nur bezüglich zulässiger Unterklassen; Klassenäquivalenz nur für Ausdrücke, die gleichzeitig die Kriterien für Unter- und Überklassen erfüllen; keine Einschränkungen hinsichtlich `owl:sameAs`
- einige Einschränkung hinsichtlich verwendbarer Datentypen



# OWL 2 RL: FEATURES

AIFB 

- Standard reasoning in OWL 2 RL:  
PTime-vollständig
- Regelbasierter Ansatz vereinfacht Modellierung  
und Implementierung:  
selbst “naive” Implementierungen können nützlich  
sein
- schnelle und skalierbare Inferenzmaschinen  
(z.B.: Oracle)

# Do WE REALLY NEED SO MANY OWLS?

AIFB 

- **Drei neue OWL-Profiles mit recht komplexen Beschreibungen ... warum nicht nur eine?**
- die Vereinigung von je zwei dieser drei Profile ist nicht mehr leichtgewichtig! Reasoning in QL+RL, QL+EL, RL+EL ist ExpTime-hart
- Einschränkung auf weniger Profile = Aufgabe potenziell nützlicher Kombinationen von Modellierungsmitteln
- Grundidee: Profile sind “größt-mögliche” berechnungstechnisch gutartige Teilsprachen von OWL 2 → Auswahl der passenden Sprache je nach Anwendung



# OWL IN DER PRAXIS: TOOLS



AIFB 

- Editoren (<http://semanticweb.org/wiki/Editors>)
  - der gebräuchlichste: [Protégé 4](#)
  - andere: [TopBraid Composer](#) (\$), [NeOn toolkit](#)
  - special purpose apps, besonders für leichtgewichtige Ontologien (z.B. [FOAF](#)-Editoren)
- Reasoner (<http://semanticweb.org/wiki/Reasoners>)
  - OWL DL: [Pellet](#), [HermiT](#), [FaCT++](#), [RacerPro](#) (\$)
  - OWL EL: [CEL](#), [SHER](#), [snorocket](#) (\$)
  - OWL RL: [OWLIM](#), [Jena](#), [Oracle Prime](#) (part of O 11g) (\$),
  - OWL QL: [Owlgres](#), [QuOnto](#), [Quill](#)
- viele Tools verwenden die [OWL API](#) Bibliothek (Java)

# NICHTSTANDARD-REASONING IN OWL



Es gibt mehr als Editieren und deduktives automatisches Schlussfolgern:

- **Explanation:** finde Mengen von Axiomen, die eine bestimmte Konsequenz erklären (wichtig zum Editieren und Debuggen)
- **Modularisierung:** extrahiere Teilontologien, die für bestimmte Zwecke ausreichen
- **Repair:** finde und behebe Modellierungsfehler in Ontologien (verwandt mit Explanation)
- **Least Common Subsumer:** finde den speziellsten Klassenausdruck der allgemeiner ist als eine Menge gegebener Klassenausdrücke
- **Abduction:** gegeben eine gewünschte Konsequenz, finde mögliche Input-Axiome die zu dieser Konsequenz führen würden

→ dafür gibt es implementierte Verfahren, die häufig auf Standard-Reasoning-Tools aufsetzen

# ÜBERBLICK: WICHTIGE OWL FEATURES



| Feature                      | Related OWL vocabulary                                               | FOL                                                  | DL               |
|------------------------------|----------------------------------------------------------------------|------------------------------------------------------|------------------|
| top/bottom class             | <code>owl:Thing/owl:Nothing</code>                                   | (axiomatise)                                         | $\top/\perp$     |
| Class intersection           | <code>owl:intersectionOf</code>                                      | $\wedge$                                             | $\sqcap$         |
| Class union                  | <code>owl:unionOf</code>                                             | $\vee$                                               | $\sqcup$         |
| Class complement             | <code>owl:complementOf</code>                                        | $\neg$                                               | $\neg$           |
| Enumerated class             | <code>owl:oneOf</code>                                               | (ax. with $\approx$ )                                | $\{a\}$          |
| <b>Property restrictions</b> | <code>owl:onProperty</code>                                          |                                                      |                  |
| Existential                  | <code>owl:someValueFrom</code>                                       | $\exists y \dots$                                    | $\exists R.C$    |
| Universal                    | <code>owl:allValuesFrom</code>                                       | $\forall y \dots$                                    | $\forall R.C$    |
| Min. cardinality             | <code>owl:minQualifiedCardinality</code><br><code>owl:onClass</code> | $\exists y_1 \dots y_n. \dots$                       | $\geq_n S.C$     |
| Max. cardinality             | <code>owl:maxQualifiedCardinality</code><br><code>owl:onClass</code> | $\forall y_1 \dots y_{n+1}. \dots \rightarrow \dots$ | $\leq_n S.C$     |
| Local reflexivity            | <code>owl:hasSelf</code>                                             | $R(x,x)$                                             | $\exists R.Self$ |

# ÜBERBLICK: WICHTIGE OWL FEATURES



| Feature                  | Related OWL vocabulary                | DL                  |
|--------------------------|---------------------------------------|---------------------|
| Property chain           | <code>owl:propertyChainAxiom</code>   | $\circ$             |
| Inverse                  | <code>owl:inverseOf</code>            | $R^{-}$             |
| Key                      | <code>owl:hasKey</code>               | rule, see Lecture 5 |
| Property disjointness    | <code>owl:propertyDisjointWith</code> | $\text{Dis}(R,S)$   |
| Property characteristics | <code>rdf:hasType</code>              |                     |
| Symmetric                | <code>owl:SymmetricProperty</code>    | $\text{Sym}(R)$     |
| Asymmetric               | <code>owl:AsymmetricProperty</code>   | $\text{Asy}(R)$     |
| Reflexive                | <code>owl:ReflexiveProperty</code>    | $\text{Ref}(R)$     |
| Irreflexive              | <code>owl:IrreflexiveProperty</code>  | $\text{Irr}(R)$     |
| Transitivity             | <code>owl:TransitiveProperty</code>   | $\text{Tra}(R)$     |

|             |                                 |                                         |                   |
|-------------|---------------------------------|-----------------------------------------|-------------------|
| Subclass    | <code>rdfs:subClassOf</code>    | $\forall x.C(x) \rightarrow D(x)$       | $C \sqsubseteq D$ |
| Subproperty | <code>rdfs:subPropertyOf</code> | $\forall x,y.R(x,y) \rightarrow S(x,y)$ | $R \sqsubseteq S$ |

# ZUSAMMENFASSUNG



AIFB 

- OWL: ausdrucksstarke Ontologiesprache mit praktischer Bedeutung
- Strukturell in RDF representierbar (XML oder Turtle syntax)
- verschiedene Varianten für unterschiedliche Anwendungen:
  - OWL Full verfügt über volle semantische RDF-Kompatibilität (unentscheidbar)
  - OWL DL entscheidbar aber aufwändig (N2ExpTime)
  - OWL-Profiles für leichtgewichtiges Schlussfolgern (in PTime)